Documentation: Surya's Text to Math Problem Solver using Gradio

Overview

This application is an AI-powered **Text to Math Problem Solver** built using **Gradio** and **LangChain**. It takes user-inputted mathematical questions, processes them using **Groq's LLM**, and provides **step-by-step explanations** for the solutions. The system also integrates **Wikipedia search** for additional context and explanations.

Features

- Accepts **math-related text queries**.
- **Generates numerical expressions** and assumptions if needed.
- Provides **step-by-step explanations** for problem-solving.
- Uses **Wikipedia search** for additional knowledge.
- Requires a **GROQ API key** for language model inference.
- **Interactive UI** using **Gradio**.
- Offers **real-time response** generation.

Technologies Used

- **Python**: Core programming language.
- **Gradio**: Web-based UI for easy interaction.
- **LangChain**: For handling LLM-based operations and chains.
- **Groq**: AI model inference for generating explanations.
- **WikipediaAPIWrapper**: Fetches relevant content from Wikipedia.
- **Regular Expressions (re)**: Extracts numerical expressions.
- **dotenv**: Manages API keys securely.

Workflow Explanation

1. Model Initialization

- Loads the **GROQ API key** from user input.
- Initializes **Groq's LLM** (gemma2-9b-it).

2. Tools and Functionalities

Wikipedia Search Tool

- Allows searching Wikipedia for **relevant math topics**.

Math Problem Solver

- Uses **LLMMathChain** for performing calculations.
- Extracts **numerical expressions** from input queries.
- Assumes **default values for variables** when needed.
- Generates **step-by-step explanations** of the solution.

Reasoning Tool

- Uses a custom **PromptTemplate** to guide the model.
- Provides **detailed reasoning** behind mathematical solutions.

3. Response Generation

- Combines all tools into an **AI Agent**.
- Processes the question and determines the **best approach**.
- If the model response **does not contain steps**, it **falls back** to detailed calculation logic.

User Interaction via Gradio

Inputs

- **GROQ API Key** (Required for processing).
- **Math Question** (The problem statement).

Outputs

- **Step-by-step solution** explaining the mathematical process.
- **Final computed result**.
- **Assumptions made (if any variables exist in the query)**.

Buttons & Actions

- **Find my answer** → Triggers the AI system to process the input and display results.

Error Handling & Edge Cases

- **Invalid API Key** → Displays a warning and stops execution.
- **Non-Mathematical Inputs** → Returns a warning asking for a valid math question.
- **No Numerical Expression Found** → Provides an error message stating that a valid expression was not detected.
- **Unrecognized Variables** → Assumes default values and **clearly mentions** them in the output.

Deployment & Execution

- The **Gradio app** is launched in a **web browser**.
- The share=True parameter allows external users to test the application via a **public Gradio link**.
- Works locally or can be **hosted on cloud platforms**.

Use Cases

- **Students & Learners** → Step-by-step guidance for solving math problems.
- **Tutors & Teachers** → Automated solution explanations.
- **Data Analysts & Researchers** → Quick verification of numerical expressions.
- **Developers & Engineers** → AI-powered math problem-solving assistance.

Future Improvements

- **Enhanced Equation Parsing** → Improve accuracy in extracting and solving complex formulas.
- **Graphical Representations** → Visual solutions with plots/graphs.
- **Voice Input Support** → Allow users to ask math questions verbally.
- **Expanded Knowledge Base** → Integrate more data sources beyond Wikipedia.

- **Multiple AI Models** → Use different LLMs for improved reasoning and calculation accuracy.

Conclusion

This **AI-powered math solver** bridges the gap between **text-based queries** and **step-by-step numerical solutions**. With the combination of **Groq LLM, Wikipedia search, and advanced reasoning models**, it provides a **comprehensive, interactive, and intelligent problem-solving experience** for users.

Code:

```
import gradio as gr

from langchain_groq import ChatGroq

from langchain.chains import LLMMathChain, LLMChain

from langchain.prompts import PromptTemplate

from langchain_community.utilities import WikipediaAPIWrapper

from langchain.agents.agent_types import AgentType

from langchain.agents import Tool, initialize_agent

from dotenv import load_dotenv

import re

import os

# Load environment variables

load_dotenv()

# Initialize the language model

def initialize_model(groq_api_key):

    try:
```

```
        llm = ChatGroq(model="gemma2-9b-it", groq_api_key=groq_api_key)
        return llm
    except Exception as e:
        raise Exception(f"Failed to initialize model: {e}")
# Define calculate_with_explanation globally
def calculate_with_explanation(question, llm):
    try:
        math_chain = LLMMathChain.from_llm(llm=llm)
        math_prompt = """
        You are a mathematical assistant. For the given question, provide a numerical expression (no variables) and a detailed, point-wise explanation of how to solve it. If the question contains variables, assume reasonable numerical values (e.g., k=1) and state your assumption:
        Question: {question}
        Numerical Expression: <expression>
        Description:
        - Step 1: [First step]
        - Step 2: [Second step]
        - ... [Continue as needed]
        Final Result: <result>
        """
        math_prompt_template = PromptTemplate(input_variables=["question"], template=math_prompt)
        math_explain_chain = LLMChain(llm=llm, prompt=math_prompt_template)
```

```
        explanation = math_explain_chain.run({"question": question})
        expr_match = re.search(r'Numerical Expression: (.*?)\n', explanation)
        if not expr_match:
            return f"Question: {question}\nError: No valid numerical expression provided\nDescription:\n- Step 1: Failed to parse a numerical expression\nFinal Result: N/A"
        expr = expr_match.group(1).strip()
        if re.search(r'[a-zA-Z]', expr):
            assumption = "Assumption: Any variables (e.g., k) set to 1 unless specified."
            expr = re.sub(r'[a-zA-Z]', '1', expr)
            explanation = f"{explanation}\n{assumption}"
        result = math_chain.run(expr)
        return f"{explanation}\nFinal Result: {result}"
    except Exception as e:
        return f"Question: {question}\nNumerical Expression: {expr if 'expr' in locals() else 'N/A'}\nDescription:\n- Error: {e}\nFinal Result: N/A"

# Initialize the Chat Tools
def initialize_tools(llm):
    wikipedia_wrapper = WikipediaAPIWrapper()
    wikipedia_tool = Tool(
        name="Wikipedia",
        func=wikipedia_wrapper.run,
        description="A tool for searching Wikipedia to assist with math problems"
```

```
)

# Math tool using the global calculate_with_explanation

calculator = Tool(

    name="Calculator",

    func=lambda q: calculate_with_explanation(q, llm),  # Pass llm to the function

    description="Solves math questions with step-by-step explanations."

)

# Define the reasoning prompt

prompt = """

You are an agent tasked with solving the user's mathematical question. Provide a numerical expression (no variables) and a detailed, point-wise explanation. If variables are present, assume reasonable values (e.g., k=1) and note the assumption:

Question: {question}

Numerical Expression: <expression>

Description:

- Step 1: [First step]

- Step 2: [Second step]

- ... [Continue as needed]

Final Result: <result>

"""

prompt_template = PromptTemplate(input_variables=["question"], template=prompt)

try:
```

```
        chain = LLMChain(llm=llm, prompt=prompt_template)
        reasoning_tool = Tool(
            name="Reasoning",
            func=chain.run,
            description="Answers math questions with step-by-step explanations."
        )
    except Exception as e:
        raise Exception(f"Reasoning tool setup failed: {e}")
    return [wikipedia_tool, calculator, reasoning_tool]
# Function to generate the response
def generate_response(user_question, groq_api_key):
    try:
        llm = initialize_model(groq_api_key)
        tools = initialize_tools(llm)
        assistant_agent = initialize_agent(
            tools=tools,
            llm=llm,
            agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
            verbose=False,
            handle_parsing_errors=True,
        )
        response = assistant_agent.invoke({'input': user_question})
```

```
        resp = response['output'] if isinstance(response, dict) and 'output' in response else str(response)
        if "Description:" not in resp:
            return calculate_with_explanation(user_question, llm)  # Use global function with llm
        return resp
    except Exception as e:
        return f"Error: {e}"

# Gradio Interface
def gradio_interface(question, api_key):
    if not api_key:
        return "Please provide a GROQ API key."
    if not question or not question.strip():
        return "Please enter a question."
    return generate_response(question, api_key)

# Create Gradio app
with gr.Blocks(title="Surya's Text to Math Problem Solver") as demo:
    gr.Markdown("# Text to Math Problem Solver")
    gr.Markdown("Enter your math question and GROQ API key to get a step-by-step solution!")
    api_key_input = gr.Textbox(label="GROQ API Key", type="password", placeholder="Enter your GROQ API key here")
    question_input = gr.Textbox(label="Enter your Question", placeholder="e.g., What is 5 + 3 * 2?")
    output = gr.Textbox(label="Response", interactive=False)
    submit_button = gr.Button("Find my answer")
```

```
    submit_button.click(
        fn=gradio_interface,
        inputs=[question_input, api_key_input],
        outputs=output
    )
# Launch the app
demo.launch(share=True)
```